Exam : AD0-E716

Title : Adobe Commerce Developer Expert

https://www.passcert.com/AD0-E716.html

1.An Adobe Commerce developer has added an iframe and included a JavaScript library from an external domain to the website. After that, they found the following error in the console: Refused to frame [URL] because it violates the Content Security Policy directive.
In order to fix this error, what would be the correct policy ids to add to the csp_whitelist.xml file?
A. frame-src and script-src
B. default-src and object-src
C. frame-ancestors and connect-src
Answer: C
Explanation:
The frame-ancestors directive specifies the domains that are allowed to embed the current page in an iframe. The connect-src directive specifies the domains that are allowed to be loaded by the current page through a <script> tag or XMLHttpRequest.
In this case, the developer has added an iframe that embeds a page from an external domain. The Content Security Policy (CSP) is preventing the iframe from being loaded because the domain of the external page is not listed in the frame-ancestors directive.
To fix this error, the developer needs to add the domain of the external page to the frame-ancestors directive. They can do this by adding the following line to the csp_whitelist.xml file:
<frame-ancestors>https://www.example.com</frame-ancestors>

2.An Adobe Commerce Developer is tasked with creating a custom form which submits its data to a frontend controller They have decided to create an action and have implemented the \Magento\Framework\App\Action\HttpPostActioninterface class, but are not seeing the data being persisted in the database, and an error message is being shown on the frontend after submission.
After debugging and ensuring that the data persistence logic is correct, what may be cause and solution to this?
A. Magento does not allow POST requests to a frontend controller, therefore, the submission functionality will need to be rewritten as an API endpoint.
B. The developer forgot to implement a validatePostDataQ method in their action. They should implement this method: all non-validated POST data gets stripped out of the request and an error is thrown.
C. Form key validation runs on all non-AJAX POST requests, the developer needs to add the for_key to their requests.
Answer: C
Explanation:
According to the Magento Stack Exchange answer, form key validation is a security feature that prevents CSRF attacks by checking if the form key in the request matches the one generated by Magento. If the developer does not include the form_key in their custom form, the validation will fail and an error will be shown. Therefore, the developer needs to add the form_key to their requests by using <?= $block->getBlockHtml (‘formkey’) ?> in their template file.
Reference: https://magento.stackexchange.com/questions/95171/magento-2-form-validation

3.An Adobe Commerce developer is working on a module to manage custom brand entities and wants to replicate the following SQL query using SearchCriteria:
A)

```
$filter1->setField('featured')
    ->setValue(1)
    ->setConditionType('eq');

$filter2->setField('logo_image')
    ->setConditionType('notnull');

$filterGroup1->setFilters([$filter1, $filter2]);

$filter3->setField('enabled')
    ->setValue(1)
    ->setConditionType('eq');

$filterGroup2->setFilters([$filter3]);

$searchCriteria->setFilterGroups([$filterGroup1, $filterGroup2]);
```

B)

```
$filter1->setField('featured')
    ->setValue(1)
    ->setConditionType('eq');

$filter2->setField('logo_image')
    ->setConditionType('notnull');

$filter3->setField('enabled')
    ->setValue(1)
    ->setConditionType('eq');

$searchCriteria->setFilter($filter3)
    ->setOrFilter([$filter1, $filter2]);
```

C)

```
$filter1->setField('enabled')
    ->setValue(1)
    ->setConditionType('eq');

$searchCriteria->setFilter($filter1);
$filter2->condition('featured = ? OR logo_image ?, [1, 'IS NOT NULL'])
$searchCriteria->setSelectFields($filter2);
```

A. Option A

B. Option B

C. Option C

Answer: B

Explanation:

The following SearchCriteria query will replicate the SQL query:

$searchCriteria = new \Magento\Framework\Api\SearchCriteriaBuilder();

$searchCriteria->addFilter('name', 'Brand 1', 'eq'); $searchCriteria->addFilter('status', 1, 'eq');

$brandCollection = $this->brandRepository->getList($searchCriteria);

4.The di. xml file of a module attaches two plugins for the class Action.

The PluginA has the methods: beforeDispatch, aroundDispatch, afterDispatch.
The PluginB has the methods: beforeDispatch, afterDispatch.

```
<config>
    <type name="Magento\Framework\App\Action\Action">
        <plugin name="vendor_module_plugina" type="Vendor\Module\Plugin\PluginA" sortOrder="10" />
        <plugin name="vendor_module_pluginb" type="Vendor\Module\Plugin\PluginB" sortOrder="20" />
    </type>
</config>
```

The around plugin code is:

```
class PluginA
{
    public function aroundDispatch(\Magento\Framework\App\Action\Action $subject, $next, $request)
    {
        // custom code
        return $request;
    }
}
```

What would be the plugin execution order?

A)

```
PluginA::beforeDispatch()

PluginA::aroundDispatch()

PluginB::beforeDispatch()

Action::dispatch()

PluginB: afterDispatch()

PluginA::aroundDispatch()
```

B)

```
PluginA::beforeDispatch()

PluginA::aroundDispatch()

PluginA: afterDispatch()


PluginA::beforeDispatch()

PluginA::aroundDispatch()

PluginB::beforeDispatch()
```

C)

```
Action::dispatch()

PluginB: afterDispatch()

PluginA::aroundDispatch()

PluginA::afterDispatch()
```

A. Option A
B. Option B
C. Option C

Answer: C

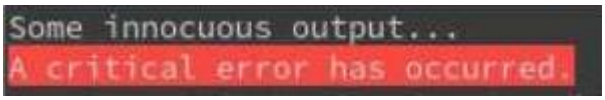
Explanation:

The plugin execution order is as follows:

PluginA::beforeDispatch()
PluginB::beforeDispatch()
PluginA::aroundDispatch()
The code in the around plugin
PluginB::afterDispatch()
PluginA::afterDispatch()
The aroundDispatch() method is executed in a separate scope, so the code in the around plugin will be executed after the beforeDispatch() methods of both plugins, but before the afterDispatch() methods of both plugins.
Here is a diagram that shows the plugin execution order:
PluginA
beforeDispatch()
aroundDispatch()
afterDispatch()
PluginB
beforeDispatch()
afterDispatch()

5.An Adobe Commerce developer adds a new extension attribute to add an array of values to the invoices that are fetched through the APIs.
After a while, their technical manager reviews their work and notices something wrong with the extension_attributes. xml file that the developer created in their module: What is the problem with this xml snippet?
A. The extension attribute references the wrong interface, it should have referenced the Magento\saies\Api\data\invoiceinterface.
B. The extension attribute references the repository instead of the interface it implements (Magento\saies\Api\invoiceRepositorymterface).
C. The type is wrong, string [] should be replaced with array.
Answer: B
Explanation:
The extension attribute is referencing the repository instead of the interface it implements.
The correct XML snippet should be:
XML
<extension_attributes>
<attribute code="custom_values" type="string[]"
group="General"
translate="true">
<description>This attribute stores an array of custom values for the invoice.</description>
<source_model>Magento\Sales\Api\Data\InvoiceInterface</source_model>
</attribute>
</extension_attributes>
The source_model attribute specifies the interface that the extension attribute is associated with. In this case, the extension attribute is associated with the Magento\Sales\Api\Data\InvoiceInterface interface.

6.An Adobe Commerce developer is creating a new console command to perform a complex task with a lot of potential terminal output.
If an error occurs, they want to provide a message that has higher visibility than some of the other content that may be appearing, so they want to ensure it is highlighted in red (as seen in the screenshot):



How can they customize the appearance of this message?
A. Call the setDecorationType(Stype) method On the Symfony\Console\Output\OutputInterface Object before Calling writeln().
B. Wrap the output content in tags like <error>, <info>, or <comment>.
C. Throw a new commandException with the desired message passed as an argument.
Answer: A
Explanation:
The developer can customize the appearance of the error message by calling the setDecorationType() method on the Symfony\Console\Output\OutputInterface object before calling writeln(). The setDecorationType() method takes a single argument, which is the type of decoration that the developer wants to use. In this case, the developer wants to use the STYPE_ERROR decoration, which will highlight the message in red.
Here is an example of how to customize the appearance of the error message:
$output = new Symfony\Console\Output\ConsoleOutput();
$output->setDecorationType(Symfony\Console\Output\OutputInterface::STYPE_ERROR);
$output->writeln('This is an error message.');
The output of this code will be an error message that is highlighted in red.

7.An Adobe Commerce developer is being tasked with creating a new cron job to run a method that has already been written.
What are the minimally required steps to accomplish this?
A. Create a crontab.xmi file and a new system configuration in system.xmi for the schedule.
B. Create crontab.xmi and cron_groups.xmi files to assign the new job to a cron group.
C. Create a crontab.xmi file and set a schedule for the new cron job.
Answer: C
Explanation:
According to the Configure and run cron guide for Magento 2 developers, the crontab.xmi file is used to declare and configure cron jobs for a module. The file should specify the name, instance, method and schedule of the cron job. Therefore, creating a crontab.xmi file and setting a schedule for the new cron job are the minimally required steps to accomplish this task.
Reference: https://devdocs.magento.com/guides/v2.3/config-guide/cli/config-cli-subcommands-cron.html

8.Which hashing algorithm will Adobe Commerce choose to hash customer passwords?
A. If the Sodium extension is installed, SHA256 will be chosen, otherwise MD5 will be used as the Magento default hashing algorithm.
B. If the Sodium extension is installed, Argon 2ID13 will be chosen, otherwise SHA256 will be used as the Magento default hashing algorithm.

C. It does not matter if the Sodium extension is installed or not, the Magento hashing default algorithm will be SHA256.

Answer: B

Explanation:

If the Sodium extension is installed, Argon 2ID13 will be chosen as the Magento default hashing algorithm. Otherwise, SHA256 will be used.

The Sodium extension is a PHP extension that provides cryptographic functions. Argon 2ID13 is a password hashing algorithm that is considered to be more secure than SHA256.

If the Sodium extension is installed, Magento will use Argon 2ID13 as the default hashing algorithm for customer passwords. If the Sodium extension is not installed, Magento will use SHA256 as the default hashing algorithm.

9.An Adobe Commerce developer is developing a custom module. As part of their implementation they have decided that all instances of their Custom\Module\Model\Example class should receive a new instance of Magento\Filesystem\Adapter\Local.

How would the developer achieve this using di. xml?

A)

```
<type name="Custom\Module\Model\Example">
    <arguments>
        <argument name="adapter" xsi:type="object" shared="false">Magento\Filesystem\Adapter\Local</argument>
    </arguments>
</type>
```

B)

```
<type name="Custom\Module\Model\Example">
    <arguments>
        <argument name="adapter" xsi:type="object" singleton="false">Magento\Filesystem\Adapter\Local</argument>
    </arguments>
</type>
```

C)

```
<type name="Custom\Module\Model\Example">
    <arguments>
        <argument name="adapter" xsi:type="object" transient="true">Magento\Filesystem\Adapter\Local</argument>
    </arguments>
</type>
```

A. Option A

B. Option B

C. Option C

Answer: B

Explanation:

The developer can achieve this by adding the following configuration to their di.xml file:

XML

<config>

<component name="Custom\Module\Model\Example"

factory="Custom\Module\Model\ExampleFactory">

<arguments>

<argument name="filesystemAdapter" type="Magento\Filesystem\Adapter\Local" /> </arguments>

</component>

</config>

This configuration will ensure that all instances of the Custom\Module\Model\Example class will receive a new instance of the Magento\Filesystem\Adapter\Local class.

10.An Adobe Commerce developer has been tasked with applying a pricing adjustment to products on the website. The adjustments come from a database table. In this case, catalog price rules do not work. They created a plugin for getPrice on the price model, but the layered navigation is still displaying the old price.
How can this be resolved?
A. Create an implementation for \Magento\Catalog\Hodel\Product\PriceModifierInterf ace.
B. Create an after plugin On \Magento\Catalog\Api\Data\BasePriceInterface:: getPrice.
C. Create a plugin for\Magento\Catalog\Model\Indexer\Product\Price::executeRow.
Answer: C
Explanation:
The developer can resolve this issue by creating a plugin for the Magento\Catalog\Model\Indexer\Product\Price::executeRow() method. This method is responsible for updating the product price index.
The plugin can be used to add the pricing adjustment from the database to the product price index. Once the product price index is updated, the layered navigation will display the correct price.
Here is an example of a plugin for the executeRow() method:
PHP

```
class MyPlugin
{
public function executeRow(
\Magento\Catalog\Model\Indexer\Product\Price $subject,
\Magento\Catalog\Model\Product $product,
array $data
) {
$adjustment = $this->getAdjustment($product);
$product->setPrice($product->getPrice() + $adjustment);
}
private function getAdjustment(Product $product)
{
$adjustment = $product->getData('adjustment');
if (!is_numeric($adjustment)) {
return 0;
}
return $adjustment;
}
}
```

This plugin will add the adjustment data from the product to the product price index. Once the product price index is updated, the layered navigation will display the correct price.

11.An Adobe Commerce developer is writing an integration test. They checked some Integration Tests for Magento core modules for reference and noticed that they use data fixtures initialized by adding annotations to test classes. For example:

```
/**
 * @magentoDataFixture Magento/AdminNotification/_files/notifications.php
 */
```

The developer wants to add their own fixture to test a MyVendor_MyModule they created.
Which steps will make this possible?
A. 1. Create a PHP file with the fixture data inside their own module in [module dir]/Test/integration/_fiies/my_fixture.php.
2. Add the following annotation to the test method:

```
/**
 * @magentoDataFixture MyVendor_MyModule::Test/Integration/_files/my_fixture.php
 */
```

B. 1. Create a PHP file With the fixture data in [magento root dir]/dev/tests/integration/testsuite/MyVendor/MyModule/_files/my_fixture.php.
2. Add the following annotation to the test method:

```
/**
 * @magentoDataFixture MyVendor_MyModule::_files/my_fixture.php
 */
```

C. 1. Create a PHP file with the fixture data inside their own module in [module dir]/Test/integration/_f iies/my_f ixture.php.
2. Add the following annotation to the test method:

```
/**
 * @magentoDataFixture MyVendor/MyModule/_files/my_fixture.php
 */
```

A. Option A
B. Option B
C. Option C
Answer: B
Explanation:
To add a custom fixture to test a MyVendor_MyModule, the developer needs to do the following: Create a PHP file with the fixture data in [magento root dir]/dev/tests/integration/testsuite/MyVendor/MyModule/_files/my_fixture.php. Add the following annotation to the test method:
@magentoDataFixture(
'testsuite/MyVendor/MyModule/_files/my_fixture.php'
)
This will tell Magento to load the fixture data from the my_fixture.php file before the test method is executed.